数据降维（Dimensionality Reduction）

1.1降维的基本概念

1.1.1数据降维的定义

数据降维是一种用来减少数据矩阵中特征数量的方法。它在保证数据主要信息尽可能不丢失的情况下，将高维数据映射到低维空间，以减少数据的维度。降维的主要目标是**降低计算复杂度**、**去除冗余信息**、**减少噪声**，从而提高模型的泛化能力和计算效率。

1.1.2数据降维的作用

①降低计算成本，提高处理效率

高维数据往往伴随着**较高的计算复杂度**，无论是在**存储**还是**计算**上都可能带来较大的资源消耗。降维可以减少数据的维度，使得机器学习算法运行更快，尤其对于**大规模数据集**，降维能显著提升模型训练和推理的速度。

② 缓解“维度灾难”问题

在高维空间中，数据点往往变得**稀疏**，欧式距离等传统度量方式可能失效，使得模型难以学习有效的模式。这种现象被称为**维度灾难**，降维能够减少数据的冗余信息，使数据在低维空间中更加集中，进而提高模型的学习能力。

③去除冗余特征，提升模型泛化能力

高维数据中可能存在大量**相关性高的特征**，会导致模型的过拟合问题。降维方法（如 PCA、LDA）能够提取最具代表性的特征，去除冗余信息，使模型更加简洁，进而提高模型的泛化能力。

④降噪，提高数据质量

数据集中可能包含**噪声特征**，如果直接用于训练模型，可能会影响模型的准确性。降维方法可以通过提取主要信息、去除无关特征，减少噪声的影响，从而提高数据质量，使模型学习到更加稳定的模式。

⑤便于数据可视化和解释

对于高维数据，人类无法直观理解其模式。通过降维（如 PCA、t-SNE、UMAP）将数据映射到 **2D 或 3D 空间**，可以更直观地观察数据的分布、类别关系和潜在结构，有助于数据探索和模式发现。

⑥提高模型的可解释性

在很多机器学习任务（如医学、金融）中，可解释性至关重要。如果数据维度过高，模型的决策过程可能变得复杂且难以理解。降维能够保留最关键的信息，使得模型更加透明，从而提高结果的可解释性。

1.1.3降维算法的分类

降维算法可以根据数据映射方式分为**线性降维方法**和**非线性降维方法**，两者的主要区别在于是否假设数据的低维结构可以通过线性变换表示。

线性降维方法

线性降维方法适用于数据的主要特征可以通过线性变换提取的情况，通常依赖矩阵分解或统计特性来降低维度。常见方法包括：

- **奇异值分解（SVD, Singular Value Decomposition）**：将数据矩阵分解为三个子矩阵，其中包含数据的最重要特征，广泛应用于数据降维、压缩和推荐系统。
- **主成分分析（PCA, Principal Component Analysis）**：利用特征值分解或奇异值分解，找到数据方差最大的方向，将数据投影到低维空间，适用于特征相关的数据。
- **因子分析（FA, Factor Analysis）**：假设观测数据由多个潜在因子线性组合而成，通过建模因子结构来进行降维，常用于心理学、社会科学和经济数据分析。
- **线性判别分析（LDA, Linear Discriminant Analysis）**：一种监督学习方法，利用类别信息**最大化类间方差，最小化类内方差**，增强数据的可分性，主要用于分类任务。

非线性降维方法

当数据的结构呈现复杂的**非线性流形**时，线性方法可能无法有效表示数据特征，此时需要使用非线性降维方法。这些方法通常基于流形学习或概率分布来保持数据的局部或全局几何结构。

- **多维尺度变换（MDS, Multidimensional Scaling）**：基于样本间距离矩阵，寻找低维空间中的嵌入，使得样本间的**相对距离**尽可能接近原始高维空间。
- **等距特征映射（IsoMap, Isometric Mapping）**：利用**测地距离**而非欧氏距离来保留数据的全局几何结构，适用于嵌入在非线性流形中的数据降维。
- **t-分布随机邻域嵌入（t-SNE, t-distributed Stochastic Neighbor Embedding）**：通过**最小化高维空间和低维空间的概率分布差异**，保持数据的局部结构，广泛用于高维数据可视化。
- **统一流形逼近与投影（UMAP, Uniform Manifold Approximation and Projection）**：利用拓扑结构和流形学习对数据进行降维，比 t-SNE 计算更快，并能更好地保留全局和局部结构，适用于高维数据的可视化和聚类。

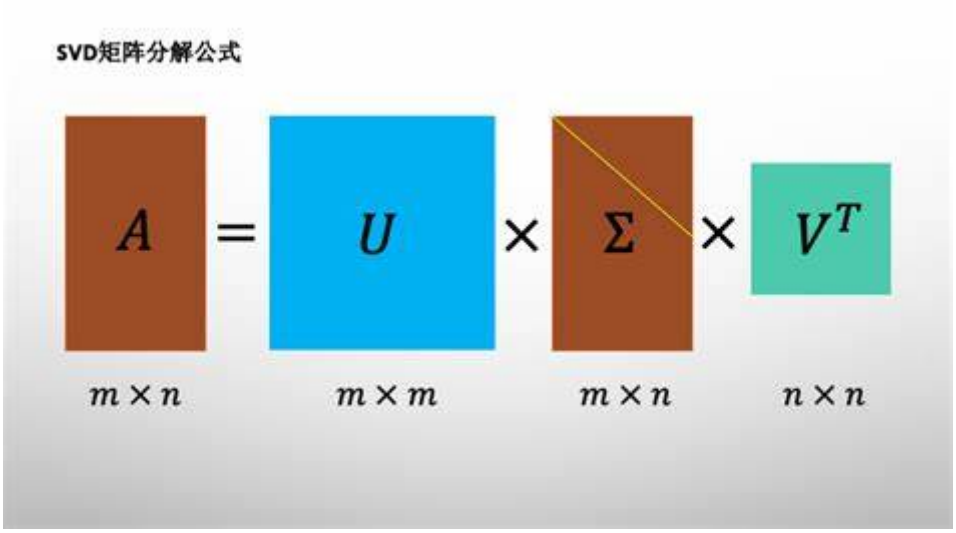
1.2奇异值分解

奇异值分解（SVD）是一种用于矩阵分解的数学工具，常用于**降维、特征提取、数据压缩、推荐系统**等场景。它能将任意一个$m \times n$维矩阵 A 分解为三个矩阵的乘积：

$$A = U\Sigma V^T$$

其中：

- U是一个$m \times m$的正交矩阵（列向量是左奇异向量）。
- Σ是一个$m \times n$的对角矩阵（对角线上的元素称为**奇异值**）。
- V是一个$n \times n$的正交矩阵（列向量是右奇异向量）。



SVD 在降维中的作用类似于 PCA，因为它能找到数据最重要的方向，并去除冗余信息。

1.2.1数学推导

给定一个矩阵 A，我们通过以下步骤计算 SVD 分解：

(1) 计算协方差矩阵

为了找到数据的主要方向，我们计算矩阵的**协方差矩阵**：

$$C = A^T A$$

这是一个n×n的对称矩阵。

(2) 计算右奇异向量（V）

我们求协方差矩阵的**特征值和特征向量**：

$$A^T AV = \lambda V$$

特征向量*V*形成了矩阵的右奇异向量。

(3) 计算左奇异向量（U）

通过计算：

$$U = AV\Sigma^{-1}$$

得到左奇异向量。

(4) 计算奇异值

奇异值 σi 是协方差矩阵特征值的平方根：

$$\sigma_i = \sqrt{\lambda_i}$$

然后，我们将奇异值放入对角矩阵*Σ*中。

(5)数据降维

我们可以只保留前k个最大奇异值，对应的前k个奇异向量构成矩阵：

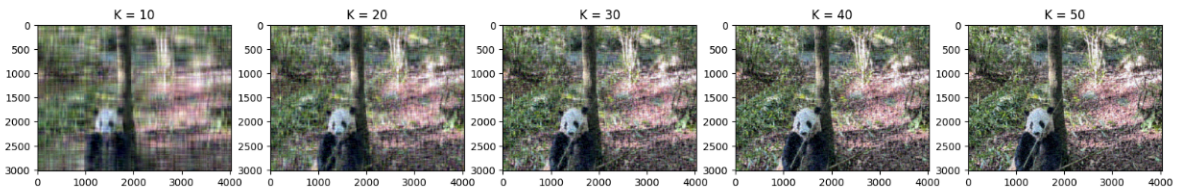
$$A_k = U_k \Sigma_k V_k^T$$

这样，我们用较低维度的矩阵近似原始矩阵A，达到降维的目的。

1.2.2实践案例：基于奇异值分解的图像压缩

下面这段代码使用**奇异值分解**对**RGB 图像**进行压缩，核心思想是利用 SVD 分解图像的每个颜色通道（R、G、B），然后仅保留前 K 个最大的奇异值来进行近似重构，从而降低存储和计算成本，同时保持较高的图像质量。

```
import numpy as np
import os
from PIL import Image
import matplotlib.pyplot as plt
# 1.定义恢复函数，由分解后的矩阵恢复到原矩阵
def restore(u, s, v, K):
    n, p = len(u), len(v[0])
    a = np.zeros((n, p))
    for k in range(K):
        uk = u[:, k].reshape(n, 1)
        vk = v[k].reshape(1, p)
        a += s[k] * np.dot(uk, vk)
    a = a.clip(0, 255)
    return np.rint(a).astype('uint8')
A = np.array(Image.open("C:/Users/HP/Desktop/大数据分析/降维/data/svd.jpg", 'r'))
# 2.对RGB图像进行奇异值分解
u_r, s_r, v_r = np.linalg.svd(A[:, :, 0])
u_g, s_g, v_g = np.linalg.svd(A[:, :, 1])
u_b, s_b, v_b = np.linalg.svd(A[:, :, 2])
# 3.保存奇异值为10，20，30，40，50的图像，并输出特征数
selected_svd_values = [10, 20, 30, 40, 50]
for k in selected_svd_values:
    R = restore(u_r, s_r, v_r, k)
    G = restore(u_g, s_g, v_g, k)
    B = restore(u_b, s_b, v_b, k)
    I = np.stack((R, G, B), axis=2)
    Image.fromarray(I).save(f'svd_{k}.jpg')
# 4.可视化重构图像
fig, axes = plt.subplots(1, 5, figsize=(20, 8))
for i, k in enumerate(selected_svd_values):
    img = Image.open(f'svd_{k}.jpg')
    axes[i].imshow(img)
    axes[i].set_title(f'K = {k}')
plt.show()
```



代码流程总结

1. **读取图像并转换为 NumPy 数组**，获取 R、G、B 三个通道的数据。
2. **对每个通道分别进行 SVD 分解**，将原始矩阵拆分为 U、S、V 三个矩阵。
3. **使用不同数量的奇异值 K 进行近似重构**，仅保留前 K 个特征信息，从而减少数据量。
4. **将重构后的 R、G、B 组合回三通道图像并保存**，用于观察不同 K 值下的压缩效果。
5. **可视化不同 K 值下的压缩图像**，直观展示 SVD 在图像压缩中的应用。

上述结果表明保留前50个奇异值的图像压缩情况较好，能够还原原始图像的大部分特征。SVD算法可以获得原始图像的压缩近似值，在减小图像尺寸的同时保留原始图像的特征信息。

1.3 主成分分析（PCA, Principal Component Analysis）

主成分分析是一种**线性降维方法**，其核心思想是通过**正交变换**，将高维数据投影到一个新的低维空间，同时**尽可能保留数据的方差信息。**

1.3.1核心思想

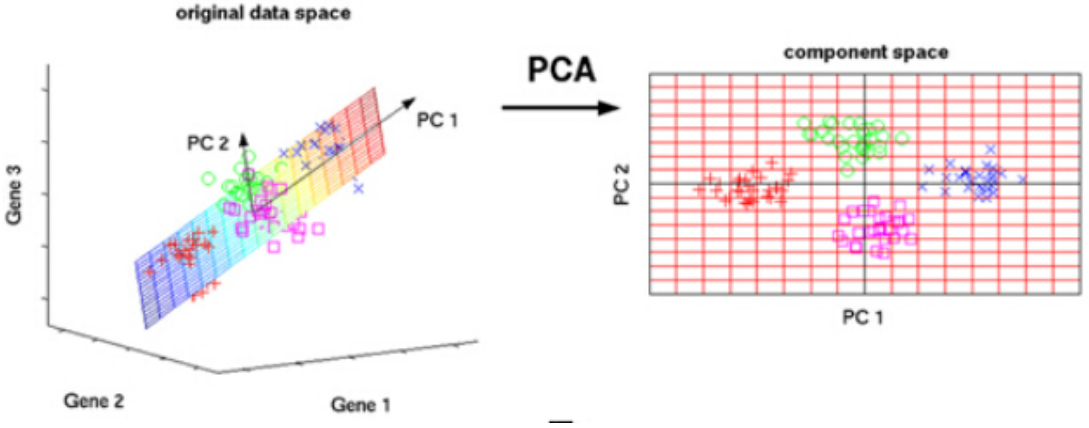
PCA 通过寻找数据中**方差最大**的方向，构造一组新的**正交基（主成分）**，并按方差大小排序，选择前几个主成分来近似表示原始数据。

具体来说：

1. **去均值**：将数据集进行标准化处理，使得所有特征的均值为 0，以消除不同特征尺度对分析的影响。
2. **计算协方差矩阵**：衡量不同特征之间的相关性，主成分的方向将沿着数据方差最大的方向。
3. **求解特征值和特征向量**：对协方差矩阵进行**特征值分解**，特征向量即为主成分方向，特征值表示该方向上的方差大小。
4. **降维**：选取前 k 个**特征值最大**的特征向量作为新的坐标轴，数据投影到这些主成分上，从而降低数据维度。

主成分分析的核心特点为：

- **最大方差保留原则**：PCA 选择的主成分方向是数据方差最大的方向，使得信息损失最小。
- **线性变换**：PCA 通过**正交变换**将数据从原始空间映射到新的坐标轴（主成分轴）。
- **特征解耦**：PCA 生成的主成分是**相互正交**的，减少了数据中的冗余信息。
- **无监督学习方法**：PCA 只关注数据的统计特性，而不考虑类别标签，因此是一种**无监督学习**方法。



1.3.2理论推导

1. 问题设定

设数据集

$$\mathbf{X} \in \mathbb{R}^{n \times d}$$

其中：

- *n*是样本数，
- *d*是原始数据的维度，
- 每个样本

$$\mathbf{x}_i \in \mathbb{R}^d$$

是一个*d*维向量。

我们的目标是找到一组新的**正交基**，并将数据投影到这组基向量上，同时保证投影后的数据方差最大，即信息损失最小。

2. 数据标准化

在进行 PCA 之前，我们需要对数据进行中心化（零均值化）：

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

然后对数据进行零均值化：

$$X = X - \bar{x}$$

这一步的作用是保证数据的原点对齐，使得 PCA 仅受数据的协方差信息影响，而不受绝对数值的影响。

---

3. 协方差矩阵

数据去中心化后，我们计算样本的协方差矩阵：

$$\mathbf{\Sigma} = \frac{1}{n} \mathbf{X}^T \mathbf{X}$$

其中：

- $$\mathbf{\Sigma} \in \mathbb{R}^{d \times d}$$

是一个对称的半正定矩阵，描述了各个维度特征之间的相关性

我们的目标是找到一个单位向量*w*，使得数据投影到这个方向后，方差最大。

---

4. 目标函数：最大化方差

设数据投影到单位向量*w*上后，新的投影值为：

$$z_i = \mathbf{w}^T \mathbf{x}_i$$

投影后的方差为：

$$\mathrm{Var}(z) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i)^2 = \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w}$$

我们希望找到一个单位向量 *w***最大化**方差，即：

$$\max_{\mathbf{w}} \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w}$$

同时，我们施加单位约束：

$$\|\mathbf{w}\|_2 = 1$$

---

5. 拉格朗日乘子法求解

利用拉格朗日乘子法，构造拉格朗日函数：

$$\mathcal{L}(\mathbf{w}, \lambda) = \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} - \lambda (\mathbf{w}^T \mathbf{w} - 1)$$

对*w*求导，并令导数为零：

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 2\boldsymbol{\Sigma} \mathbf{w} - 2\lambda \mathbf{w} = 0$$

整理得：

$$\boldsymbol{\Sigma} \mathbf{w} = \lambda \mathbf{w}$$

这表明*w*必须是**协方差矩阵的特征向量**，而对应的**最大特征值***λ**则表示数据沿该方向的方差。

---

6. 选取主成分

从特征分解结果中：

$$\boldsymbol{\Sigma} \mathbf{W} = \mathbf{W} \boldsymbol{\Lambda}$$

其中：

- $$\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$$

  为特征值矩阵

  $$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

- $$\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_d]$$

  为对应的特征向量矩阵。

通常，我们选择前 **k 个最大特征值对应的特征向量** 作为主成分，将数据投影到由这些主成分构成的低维空间：

$$\mathbf{X}' = \mathbf{X} \mathbf{W}_K$$

其中：

- *wk*是包含前k个主成分的矩阵，X'是降维后的数据。

1.3.3实现步骤

1. 设数据矩阵为

$$X \in \mathbb{R}^{m \times n}$$

其中*m*是样本数，*n*是特征数。

2. **去中心化（零均值化）**：

$$X_{\text{centered}} = X - \mu$$

其中*μ*是数据均值：

$$\mu = \frac{1}{m} \sum_{i=1}^{m} X_i$$

3. 计算**协方差矩阵**：

$$C = \frac{1}{m} X_{\text{centered}}^T X_{\text{centered}}$$

4. 计算协方差矩阵的**特征值和特征向量**：

$$Cv = \lambda v$$

选择前 *k*个最大的特征值对应的特征向量，形成降维变换矩阵 *W*。

5. **数据降维**：

$$X_{\text{reduced}} = X_{\text{centered}} W$$

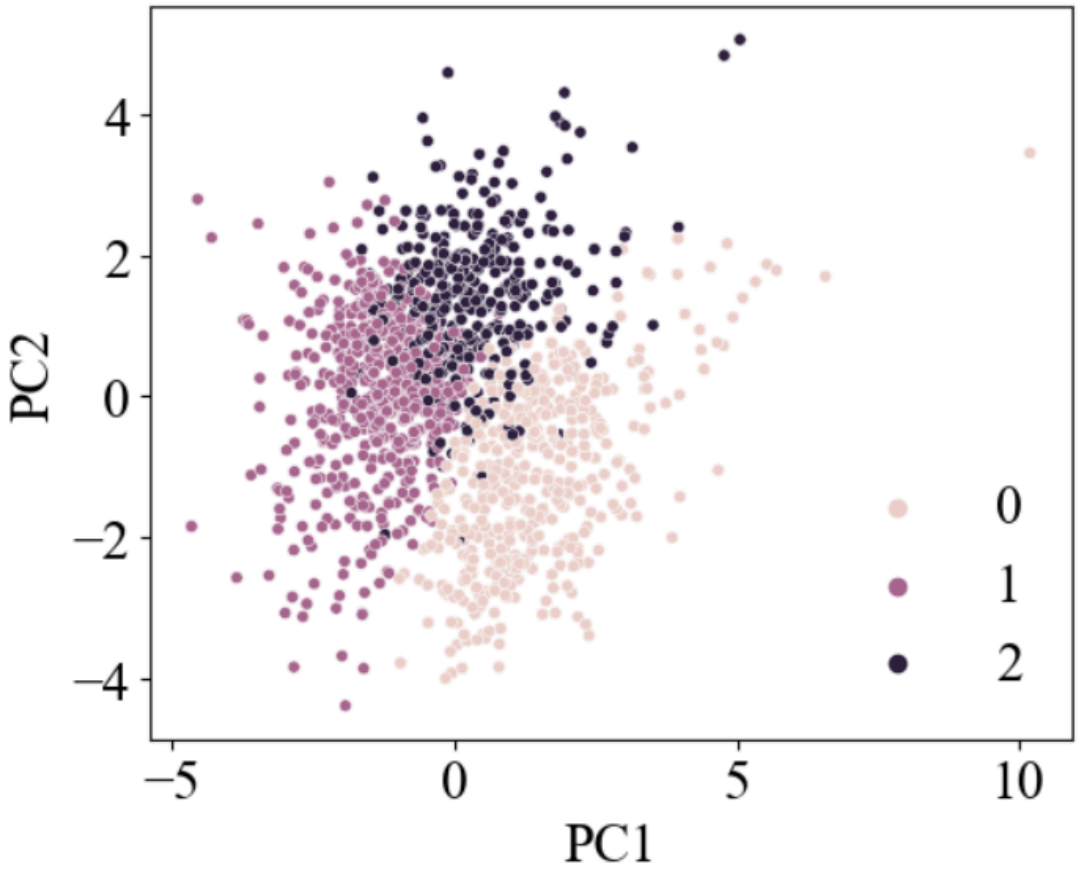
1.3.4实践案例：基于主成分分析的红酒数据集分析

fixed acic	volatile a	citric aci	residual s	chlorides	free sulfu	total sul	density	pH	sulphates	alcohol	quality
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5
7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5
11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5
7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	5
7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	7
7.8	0.58	0.02	2	0.073	9	18	0.9968	3.36	0.57	9.5	7
7.5	0.5	0.36	6.1	0.071	17	102	0.9978	3.35	0.8	10.5	5
6.7	0.58	0.08	1.8	0.097	15	65	0.9959	3.28	0.54	9.2	5
7.5	0.5	0.36	6.1	0.071	17	102	0.9978	3.35	0.8	10.5	5
5.6	0.615	0	1.6	0.089	16	59	0.9943	3.58	0.52	9.9	5
7.8	0.61	0.29	1.6	0.114	9	29	0.9974	3.26	1.56	9.1	5
8.9	0.62	0.18	3.8	0.176	52	145	0.9986	3.16	0.88	9.2	5
8.9	0.62	0.19	3.9	0.17	51	148	0.9986	3.17	0.93	9.2	5
8.5	0.28	0.56	1.8	0.092	35	103	0.9969	3.3	0.75	10.5	7
8.1	0.56	0.28	1.7	0.368	16	56	0.9968	3.11	1.28	9.3	5
7.4	0.59	0.08	4.4	0.086	6	29	0.9974	3.38	0.5	9	4
7.9	0.32	0.51	1.8	0.341	17	56	0.9969	3.04	1.08	9.2	6
8.9	0.22	0.48	1.8	0.077	29	60	0.9968	3.39	0.53	9.4	6
7.6	0.39	0.31	2.3	0.082	23	71	0.9982	3.52	0.65	9.7	5
7.9	0.43	0.21	1.6	0.106	10	37	0.9966	3.17	0.91	9.5	5
8.5	0.49	0.11	2.3	0.084	9	67	0.9968	3.17	0.53	9.4	5
6.9	0.4	0.14	2.4	0.085	21	40	0.9968	3.43	0.63	9.7	6
6.3	0.39	0.16	1.4	0.08	11	23	0.9955	3.34	0.56	9.3	5
7.6	0.41	0.24	1.8	0.08	4	11	0.9962	3.28	0.59	9.5	5
7.9	0.43	0.21	1.6	0.106	10	37	0.9966	3.17	0.91	9.5	5

红酒数据集包含了多维的数据，如度数、PH值、糖含量、酒精含量等等，此高维度的数据不利于在低维空间可视化，因此借助PCA进行数据降维，代码如下：

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
# 1.读取数据
red_wine_df = pd.read_csv("C:/Users/HP/Desktop/大数据分析/降
维/data/winequality-red.csv", index_col=0)
red_wine_df.isnull().sum()
# 2.对数据进行标准化处理
scaler = StandardScaler()
red_wine_scaled = pd.DataFrame(data=scaler.fit_transform(red_wine_df),
columns=red_wine_df.columns)
# 3.聚类
kmeans = KMeans(n_clusters=3)
clusters = kmeans.fit_predict(red_wine_scaled)
# 4.降维至二维
pca = PCA(n_components=2)
pca_2D = pca.fit_transform(red_wine_scaled)
# 将PCA结果和聚类结果合并到一个DataFrame
pca2D_df = pd.DataFrame(data=pca_2D, columns=['PC1', 'PC2'])
pca2D_df['cluster'] = clusters
# 5.可视化降维结果
plt.rcParams['font.sans-serif']=['Times New Roman']
plt.rcParams.update({'font.size': 20})
plt.figure(figsize=(6, 5))
sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=pca2D_df, s=20)
plt.tick_params(axis='both', which='major')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend(frameon=False)
plt.tight_layout()
plt.show()
```

这段代码首先读取红酒质量数据集，并对其进行 **Z-score 标准化** 以消除不同特征量纲的影响。然后，使用 **K-means 聚类** 将数据分为 3 组，以探索数据的潜在结构。接着，利用 **主成分分析（PCA）** 将数据降至二维，以便进行可视化分析。最后，通过 **Seaborn 绘制散点图**，展示降维后的数据，并根据聚类结果进行着色，从而直观地观察数据的分布情况以及不同类别的聚合趋势。



观察结果我们可以发现，红酒数据集在最大主成分（pc1）和次大主成分（pc2）上不同品种或产地的红酒样本之间存在明显的聚类。PCA算法有助于将高维的红酒特征数据转换成较低维数的表示，便于在低维空间进行可视化展示，从而更好地理解数据的结构和关系。

1.4因子分析（Factor Analysis, FA)

因子分析是一种用于**降维**和**特征提取**的统计方法，主要用于研究变量之间的潜在关系，并将多个观测变量归结为少数几个不可观测的**潜在因子**。因子分析与 **主成分分析（PCA）** 相似，但其核心目标是解释**变量的共性**，而不是仅仅减少维度。

1.4.1基本思想

因子分析是用较少几个因子表示原始数据的大部分信息的一种线性降维方法。在一组高维观测变量中，挖掘出少数几个不可直接观测的**潜在因子**，这些因子能够解释变量之间的**相关性和共性结构**，从而减少数据维度的同时，保留变量间的核心信息，使得复杂的数据更具解释性和可分析性。

1.4.2数学原理推导

① 因子模型假设

设

$$\mathbf{x} = (x_1, x_2, \ldots, x_p)^T$$

是一个*p*维观测变量向量，我们假设它由*m*个潜在因子

$$\mathbf{f} = (f_1, f_2, \ldots, f_m)^T$$

线性组合而成，加上噪声项 ϵ:

$$\mathbf{x} = \mathbf{A}\mathbf{f} + \epsilon$$

其中：

- $\mathbf{A} \in \mathbb{R}^{p \times m}$

  是**因子载荷矩阵**，表示因子对观测变量的影响。

- $\mathbf{f} \in \mathbb{R}^{m}$

  是*m*维的**公共因子**，它们解释了变量间的相关性。

- $\epsilon \in \mathbb{R}^{p}$

  是**独特因子**，它们是变量的特有部分，且假设两两不相关。

②协方差矩阵分解

假设：

- $\mathbb{E}(\mathbf{f}) = 0$

  即因子均值为 0；

- $\mathrm{Cov}(\mathbf{f}) = \mathbf{I}$

  即因子是标准化的且相互不相关；

- $\mathrm{Cov}(\epsilon) = \mathbf{\Psi} = \mathrm{diag}(\psi_1, \ldots, \psi_p)$

  即误差项互不相关。

则观测变量的协方差矩阵可以表示为：

$$\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^T + \mathbf{\Psi}$$

其中：

- $\mathbf{A}\mathbf{A}^T$

  代表公共因子贡献的方差信息；

- Ψ代表误差方差，表示变量的特有部分。

目标：估计因子载荷矩阵A和误差方差Ψ。

1.4.3 因子分析的基本步骤

①选择因子个数

- 常用方法：
  1. **碎石图**：选择拐点处的主因子数。
  2. **累计方差贡献率**：通常选取累计方差贡献率达到 **85%~95%** 的因子数。

②因子载荷矩阵估计

- **最大似然估计**：通过最优化方法求解A和Ψ。
- **主因子法：基于特征值分解求解因子载荷。

③因子旋转

因子分析得到的因子可能难以解释，因此常采用**因子旋转**，使得因子载荷矩阵更具可解释性：

- **正交旋转**：保持因子正交，最大化因子方差，使变量尽可能依赖于少数因子。
- **斜交旋转：允许因子之间相关，提高解释能力。

④因子得分计算

得到因子载荷后，我们可以计算每个样本在各个因子上的得分：

$$\mathbf{f} = (\mathbf{A}^T \mathbf{\Psi}^{-1} \mathbf{A})^{-1} \mathbf{A}^T \mathbf{\Psi}^{-1} \mathbf{x}$$

这些因子得分可以用于后续的聚类分析、回归建模等任务。

1.4.4实践案例：基于因子分析的人格特征潜在因子挖掘

本案列使用FA算法对人格特征数据集中的潜在因子进行分析，选取数据集是国际人格测验项目库中提取的2800个受试者的人格测验报告。

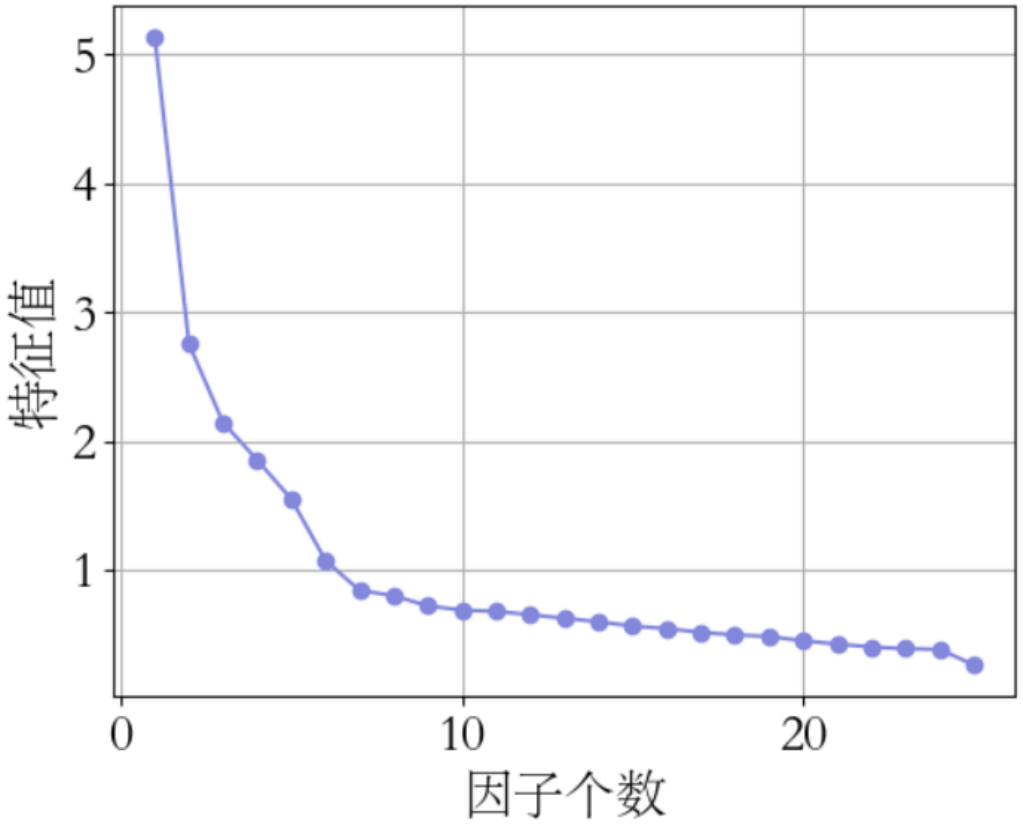
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from factor_analyzer import FactorAnalyzer
from factor_analyzer.factor_analyzer import
calculate_bartlett_sphericity,calculate_kmo
df = pd.read_csv("C:/Users/HP/Desktop/大数据分析/降维/data/character.csv",
index_col=0).reset_index(drop=True)
#1.数据处理，去掉无效字段与空值
df.drop(["gender","education","age"],axis=1,inplace=True)
df.dropna(inplace=True)
#2.进行适用性检测
chi_square_value, p_value = calculate_bartlett_sphericity(df)
print(f'p值：{p_value}')
kmo_all,kmo_model=calculate_kmo(df)
print(f'KMO：{kmo_model}')
#3.调用因子分析算法拟合数据
fa = FactorAnalyzer(25,rotation='varimax')
fa.fit(df)
#4.求解特征值ev、特征向量v
ev,v=fa.get_eigenvalues()
#5.通过绘制碎石图确定因子个数
plt.rcParams['font.family'] = ['sans-serif']
plt.rcParams['font.sans-serif'] = ['STSong']
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams.update({'font.size': 20})
plt.figure(figsize=(6, 5))
plt.plot(range(1, df.shape[1] + 1), ev, marker='o', color='#8389E0')
plt.xlabel("因子个数")
plt.ylabel("特征值")
```

```
plt.grid()
plt.tight_layout()
plt.show()
#6.选择6个因子构建模型，指定矩阵旋转方式为varimax，实现方差最大化
fa_six = fa = FactorAnalyzer(6,rotation='varimax')
fa_six.fit(df)
#7.求解因子载荷矩阵
fa_six.loadings_
#8.将原始数据用6个因子进行描述
pd.DataFrame(fa_six.loadings_,index=df.columns)
df1=pd.DataFrame(fa_six.transform(df))
df2=pd.DataFrame(fa.get_factor_variance(),index=
['variance','proportional_variance','cumulative_variances'], columns=
[f"factor{x}" for x in range(1,7)])
print(round(df1,3))
print(round(df2,3))
```

这段代码利用因子分析对人格特征数据进行降维，以挖掘数据背后的潜在因子。首先，对数据进行预处理，去除无效字段和缺失值。接着，使用 **Bartlett 球形检验和 KMO 检验**评估数据是否适合因子分析。然后，利用因子分析方法拟合数据，并计算特征值，绘制碎石图以确定最佳因子个数。随后，选取 6 个因子，并采用 **Varimax 旋转**使因子结构更具可解释性。最终，计算因子载荷矩阵，并用 6 个因子重构

数据，同时输出各因子的方差贡献情况，以便进一步分析数据的潜在结构。输出结果如下：

```
p值：0.0
KMO：0.8486452309468394
```



	0	1	2	3	4	5
0	-0.351	0.034	-1.300	-0.512	-1.429	-0.693
1	0.082	0.571	-0.612	-0.201	-0.243	-0.017
2	0.565	0.327	0.083	-0.824	0.210	-0.236
3	-0.232	0.071	-0.964	-0.268	-1.187	0.834
4	-0.337	0.365	-0.137	-0.798	-0.675	-0.190
...	...	...	...	...	...	...
2431	1.408	-1.159	-0.152	-0.935	0.527	-0.551
2432	0.711	0.302	-0.467	-0.568	0.924	0.677
2433	-0.166	0.703	0.763	-1.001	0.942	-0.428
2434	0.942	0.701	0.120	-2.207	0.682	-0.194
2435	-1.531	-1.424	-0.260	-1.465	0.069	-1.428

[2436 rows x 6 columns]

	factor1	factor2	factor3	factor4	factor5	factor6
variance	2.727	2.602	2.073	1.713	1.505	0.630
proportional_variance	0.109	0.104	0.083	0.069	0.060	0.025
cumulative_variances	0.109	0.213	0.296	0.365	0.425	0.450

求得p值小于0.05，KMO值为0.849（大于0.6），同时由碎石图可以看出，有6个因子的特征值大于1，因此选择因子分析的个数为6，由数据可以看出，6个因子可以解释45%的方差。因此，通过因子分析得到了6个新的特征来描述原始数据集，将原始数据转换成6个新的特征，实现数据降维。

1.5多维尺度变换（Multidimensional Scaling, MDS）

1. 5.1基本思想

多维尺度变换（MDS）是一种**非线性降维方法**，用于在**低维空间中表示高维数据**，使得数据点之间的**相对距离关系**尽可能保持不变。其核心思想是：

- 通过计算样本点之间的**相似性**，并在低维空间中寻找一种嵌入，使得降维后的数据点仍然尽可能保持原始高维空间中的距离关系。
- 多维尺度变换可以在不知道数据点具体坐标的情况下，通过对数据之间的距离矩阵进行特征值分解来获得数据的低维重构坐标，使降维后数据之间的相对距离变形最小。

1.5.2 MDS 的数学原理推导

①问题定义

设有*n*个样本点，原始数据在高维空间

$$\mathbb{R}^p$$

中，目标是找到一个低维嵌入，使得样本点之间的距离关系尽可能被保留。

定义：

- **距离矩阵**

  $$\mathbf{D} = [d_{ij}]$$

  其中dij表示样本点i和j之间的距离：

  $$d_{ij}\|\mathbf{x}_i - \mathbf{x}_j\|$$

  其中，xi是样本i在高维空间中的表示。

- **低维表示**

  $$\mathbf{Y} = [y_1, y_2, \ldots, y_n] \in \mathbb{R}^{n\times q}$$

  其中 q≪p，且目标是使得在低维空间中的距离dij'尽可能接近dij。

②MDS 的目标

- **保持距离关系**：

  $$d'_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\| \approx d_{ij}$$

  其中dij'是低维空间中的欧几里得距离。

- 通过优化方法找到低维表示，使得**误差最小**，即最小化目标函数：

  $$\sum_{i<j}(d_{ij} - d'_{ij})^2 i$$

  该方法称为**经典 MDS**，它基于**应力函数**进行优化。

③计算中心化距离矩阵

由于我们希望在低维空间中保持点对之间的距离，我们首先构造**Gram 矩阵** B

$$\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{D}^{(2)}\mathbf{H}$$

其中：

- D(2)是距离矩阵的平方：

  $$D_{ij}^{(2)} = d_{ij}^2$$

  双中心化矩阵H由以下公式定义：

  $$\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$$

  其中,1是全 1 向量，I是单位矩阵。

④计算低维嵌入

- 通过 **特征值分解（Eigendecomposition）** 计算B：

  $$\mathbf{B} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$$

  其中

  - $$\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_q)$$

    是B的前 qqq 个最大特征值的对角矩阵；

  - $$\mathbf{V} = [v_1, v_2, \ldots, v_q]$$

    是对应的特征向量矩阵。

- 计算降维后的表示：

  $$\mathbf{Y} = \mathbf{V}_q\mathbf{\Lambda}_q^{1/2}$$

  其中

  $$\mathbf{\Lambda}_q^{1/2}$$

  是特征值的平方根对角矩阵。

1.5.3 MDS 的基本步骤

1. **计算样本点间距离矩阵**D。
2. **构造中心化矩阵**B并进行特征值分解。
3. **选择降维维度q**（通常依据累计方差贡献率）。
4. **计算降维后样本点的位置**

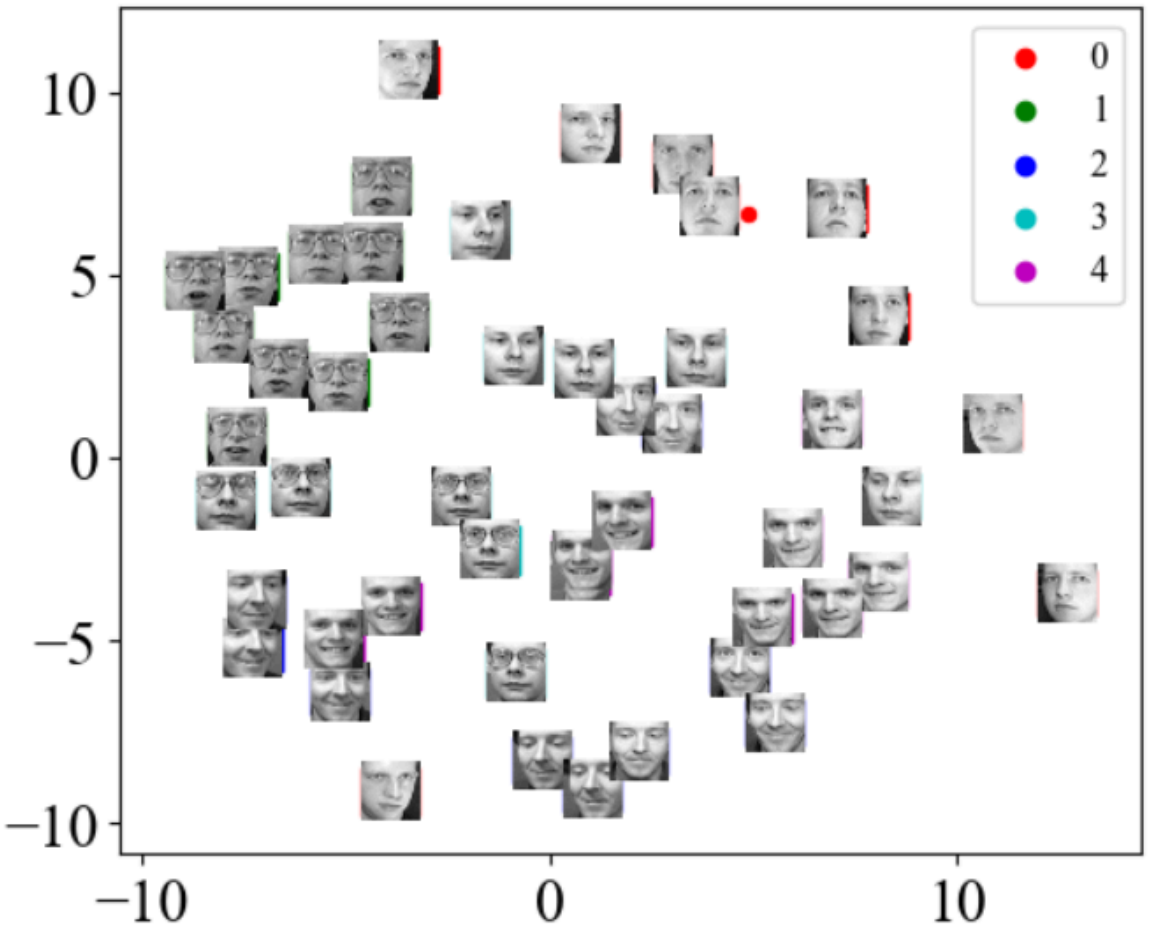
   $$\mathbf{Y} = \mathbf{V}_q\mathbf{\Lambda}_q^{1/2}$$

1.5.4实践案例：基于多维尺度变换的人脸数据集降维

本案例使用MDS算法对人脸数据集进行降维。Scikit-learn中内置数据集----Olivetti Faces共有40个志愿者的不同表情和光照条件下的人脸照片。

```
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import patches
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
from sklearn.manifold import MDS
from sklearn.metrics import euclidean_distances
from sklearn.datasets import fetch_olivetti_faces as fof

# 1.读取数据
faces = fof()
X_faces = faces.data
y_faces = faces.target
ind = y_faces < 5
X_faces = X_faces[ind, :]
y_faces = y_faces[ind]

plt.figure(figsize=(20, 10))
for i in range(50):  # 显示前50张面孔
    plt.subplot(5, 10, i + 1)
    plt.imshow(X_faces[i].reshape(64, 64), cmap='gray')
    plt.axis('off')
plt.subplots_adjust(wspace=0.2, hspace=0.2)
plt.show()

# 2.调用MDS算法对距离矩阵降维
def mapData(dist_matrix, X, y, metric, title):
    mds = MDS(metric=metric, dissimilarity='precomputed', random_state=0)
    pts = mds.fit_transform(dist_matrix)

    fig, ax = plt.subplots(figsize=(6, 5))
    color = ['r', 'g', 'b', 'c', 'm']
    sns.scatterplot(x=pts[:, 0], y=pts[:, 1], hue=y, palette=color, ax=ax)

    for x, ind in zip(X[1:], range(1, pts.shape[0])):
        im = x.reshape(64, 64)
        imagebox = OffsetImage(im, zoom=0.3, cmap=plt.cm.gray)
        i = pts[ind, 0]
        j = pts[ind, 1]
        ab = AnnotationBbox(imagebox, (i, j), frameon=False)
        ax.add_artist(ab)
        renderer = fig.canvas.get_renderer()
        bbox = ab.get_window_extent(renderer=renderer)
        w = (max(pts[:, 0]) - min(pts[:, 0])) / bbox.width
        h = (max(pts[:, 1]) - min(pts[:, 1])) / bbox.height
        rect = patches.Rectangle((i - w, j - h), w * 2, h * 2,
                                 edgecolor=color[y[ind]], linewidth=1,
fill=False)
        ax.add_patch(rect)

    ax.legend(fontsize='large', title_fontsize='large')
```

```
    plt.rcParams['font.family'] = ['sans-serif']
    plt.rcParams['font.sans-serif'] = ['Times New Roman']
    plt.rcParams.update({'font.size': 20})
    plt.tick_params(axis='both', which='major', labelsize=20)
    plt.savefig('mds1.png', dpi=600)
    plt.show()

# 3.计算欧氏距离矩阵
dist_euclid = euclidean_distances(X_faces)
mapData(dist_euclid, X_faces, y_faces, True, 'MDS')
```



代码流程如下：

1. **加载数据**：使用 `fetch_olivetti_faces()` 读取数据集，其中包含 400 张 64×64 灰度人脸图像，每个类别（共 40 个）对应 10 张不同个体的面孔。代码筛选了**前 5 类**的数据以简化分析。
2. **可视化原始人脸数据**：提取前 50 张面孔并以 5×10 的网格形式展示，每张图片通过 `imshow()` 以灰度图方式呈现，并去除坐标轴，使面孔显示更直观。
3. **MDS降维**：
   - 计算 **欧几里得距离矩阵**，衡量不同人脸图像之间的相似度。
   - 使用 **MDS（多维尺度变换）** 降维，将高维数据映射到二维平面，以便可视化。
4. **可视化：**生成带有人脸图像的 MDS 投影图

整体而言，这段代码通过 **MDS 降维 + 图像嵌入** 的方式，直观展示了不同人脸数据在低维空间中的分布，帮助理解人脸之间的相似性和类别聚类特征。

1.6等距特征映射（Isometric Mapping, Isomap）

1.6.1基本思想

等距特征映射是一种 **非线性降维** 方法，它结合了**多维尺度变换和流形学习**的思想，能够在保持**流形结构**的同时将高维数据映射到低维空间。Isomap的核心思想是：

1. **构建邻接图**：
   - 通过 **k 近邻**或 ε-邻域** 方法，确定数据点之间的连接关系，形成无向加权图。
   - 其中，边的权重是**欧几里得距离**，代表相邻点之间的实际距离。
2. **计算最短路径**：
   - 使用 **Dijkstra算法** 或**Floyd-Warshall 算法** 计算**所有数据点之间的最短路径。**
   - 这样得到的距离矩阵反映的是 **数据沿流形的测地距离**（而非欧几里得距离）。
3. **MDS 降维**：
   - 对**测地距离矩阵** 进行 **MDS（经典多维尺度分析）**，将数据映射到低维空间，同时保持原始高维空间中的测地距离关系。

1.6.2算法评价

优点：

1. **适用于非线性数据**：Isomap 通过测地距离代替欧几里得距离，更能捕捉数据的**低维流形结构。**
2. **全局最优降维**：与 LLE（局部线性嵌入）等方法不同，Isomap 通过 MDS 进行全局优化，能够更准确地还原整体结构。
3. **理论基础稳固**：Isomap 结合了图论、最短路径算法和 MDS，在流形学习方面具有扎实的数学支持。

缺点：

1. **计算复杂度高**：
   - 构建 **最短路径矩阵** 需要计算 **所有点对之间的最短路径**，计算复杂度为 **$O(N^3)$**（Floyd-Warshall）或 **$O(N^2 \log N)$**（Dijkstra）。
   - MDS 的 **特征值分解** 也需要 **$O(N^3)$** 的计算量，导致 Isomap 难以扩展到超大规模数据集。
2. **对噪声敏感**：

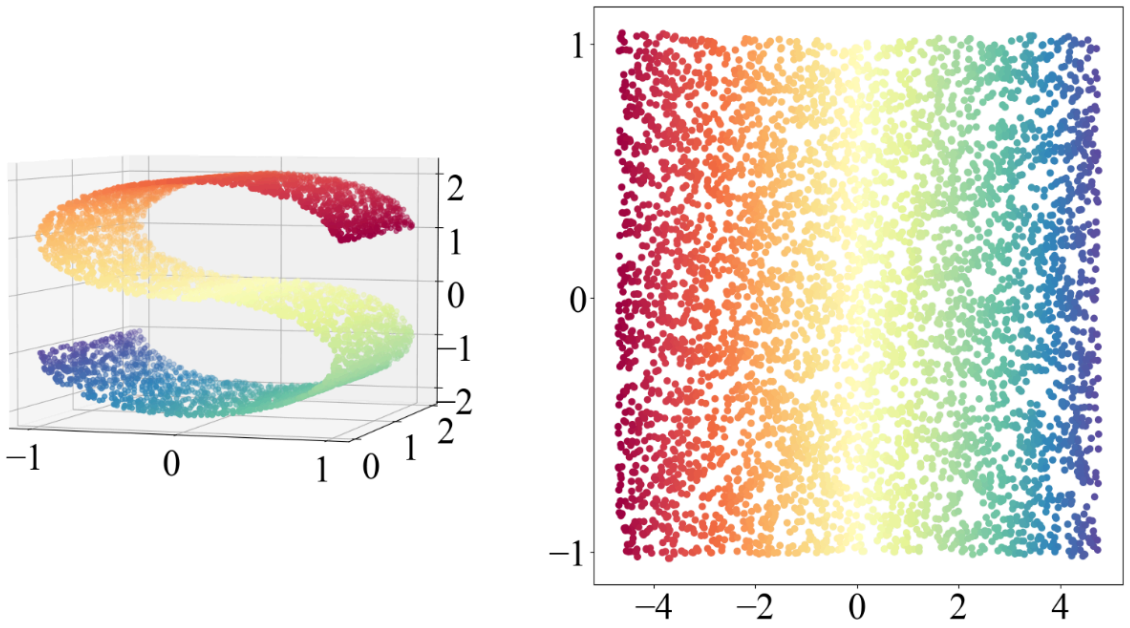
- 邻接图的构建依赖于 k 近邻或 ε-邻域，若数据分布稀疏或含有噪声，可能导致图不连通或测地距离计算错误。

3. **无法处理新数据点：
   - Isomap 需要对整个数据集重新计算最短路径和 MDS 降维，无法直接处理新样本。

1.6.3实践案例：基于等距特征映射的S状流形模型降维

本案例中，对Swiss roll数据集进行降维分析，它是一个经典的二维流形数据集，在三维空间中具有螺旋状的结构。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_s_curve
from sklearn.manifold import Isomap
from matplotlib.ticker import MaxNLocator
# 1.构建数据集
n_points = 5000
X, color = make_s_curve(n_points, random_state=0)
n_neighbors = 30
n_components = 2
plt.rcParams['font.sans-serif']=['Times New Roman']
plt.rcParams.update({'font.size': 35})
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral)
ax.view_init(4, -72)
ax.tick_params(axis='both', which='major')  # Increase axis label size
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
ax.yaxis.set_major_locator(MaxNLocator(integer=True))
ax.zaxis.set_major_locator(MaxNLocator(integer=True))
# 2.设置近邻点个数为50，将样本点映射至二维空间中
Y = Isomap(n_neighbors=n_neighbors, n_components=n_components).fit_transform(X)
ax2 = fig.add_subplot(122)
scatter = ax2.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
ax2.tick_params(axis='both', which='major')
ax2.xaxis.set_major_locator(MaxNLocator(integer=True))
ax2.yaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```



通过降维前后的对比可以发现，IsoMap算法对S状流形模型的降维效果较好，能够将不同类别的数据集进行有效分离，几乎不存在重叠。这说明IsoMap算法在保持数据之间相对距离关系的同时，也能保持不同类别之间的差异性，使数据点在降维后的空间更容易区分。

1.7 线性判别分析（LDA, Linear Discriminant Analysis）

1.7.1基本思想

线性判别分析是一种 **监督学习** 降维方法，主要用于 **分类任务**。它的核心目标是**寻找一个最佳投影方向，使不同类别的样本在低维空间中尽可能分开，同时保持同类别样本的紧凑性。**

LDA 通过构造一个 **判别超平面**，在最大化类别间方差（类间散度）的同时，最小化类别内方差（类内散度），从而提高分类的可分性。

1.7.2基本原理

LDA 主要用于分类任务，目标是**最大化类间方差，最小化类内方差**，以提高数据的可分性。

1. 设有*c*个类别，每个类别的均值为：

$$\mu_k = \frac{1}{N_k} \sum_{x_i \in C_k} x_i$$

其中*Ck*是第*k*类样本的集合，*Nk*是该类别的样本数。

2. 计算**类内散度矩阵（Sw）**：

$$S_W = \sum_{k=1}^{c} \sum_{x_i \in C_k} (x_i - \mu_k)(x_i - \mu_k)^T$$

3. 计算**类间散度矩阵（Sb）**：

$$S_B = \sum_{k=1}^{c} N_k (\mu_k - \mu)(\mu_k - \mu)^T$$

其中μ是所有样本的均值：

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \mu$$

4. 计算

$$S_W^{-1} S_B$$

的特征值和特征向量，选择最大特征值对应的特征向量进行投影。

1.7.3多类线性判别分析原理推导

设训练数据集有N个样本，d维特征，共C类，数据点表示为：

$$X = \{x_1, x_2, \ldots, x_N\}, \quad x_i \in \mathbb{R}^d$$

类别标签为

$$y_i \in \{1, 2, \ldots, C\}$$

(1) 计算类内散度矩阵

类内散度矩阵sw衡量同类样本的紧凑程度，定义为：

$$S_W = \sum_{c=1}^{C} S_W^{(c)}$$

其中，**每个类别的类内散度矩阵**：

$$SW^{(c)} = \sum_{x_i \in c} (x_i - \mu_c)(x_i - \mu_c)^T$$

- μc是类别c的均值：

$$\mu_c = \frac{1}{N_c}\sum_{x_i \in c} x_i$$

- Nc是类别c的样本数。

(2) 计算类间散度矩阵

类间散度矩阵衡量不同类别的可分离程度：

$$S_B = \sum_{c=1}^{C} N_c (\mu_c - \mu)(\mu_c - \mu)^T$$

其中：

- 是所有样本的全局均值：

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

(3) 目标函数：最大化判别能力

LDA 目标是找到一个投影矩阵W，使得：

$$W^* = \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|}$$

即在低维投影空间中，使**类间方差最大、类内方差最小。**

这个优化问题可通过**广义特征值分解**求解：

$$S_W^{-1} S_B W = \Lambda W$$

其中：

- **W是特征向量矩阵（列向量是投影方向）。**
- **Λ是对角矩阵，包含对应的特征值，反映类间和类内的可分性比率。**

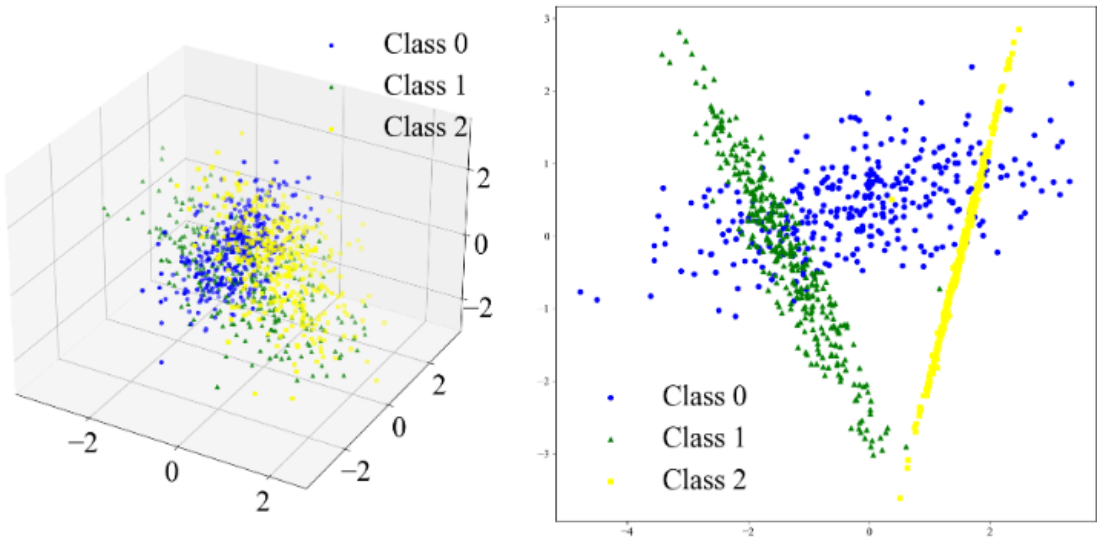
选择 **最大C−1个特征值对应的特征向量** 作为投影矩阵W，使数据降维到C−1维。

1.7.4实践案例：基于线性判别分析的三维数据集降维

本案列通过调用scikit-learn库中datasets模块的make_classification函数构建三维空间中的数据集，共1000个数据点，每个数据点一个类别，共三个类别。

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets import make_classification
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
#1.生成三维数据集
X, y = make_classification(n_samples=1000, n_features=3, n_informative=2,
n_redundant=0, n_repeated=0,
                           n_classes=3, n_clusters_per_class=1,
class_sep=0.5, random_state=10)

plt.rcParams['font.sans-serif']=['Times New Roman']
plt.rcParams.update({'font.size': 35})
fig = plt.figure(figsize=(20, 10))
#2.绘制原始数据集的散点图
ax = fig.add_subplot(121, projection='3d')
shapes = ['o', '^', 's']
colors = ['blue', 'green', 'yellow']
color_mapping = {0: colors[0], 1: colors[1], 2: colors[2]}
for i in range(3):
    markers = shapes[i]
    scatter_colors = color_mapping[i]
    ax.scatter(X[y == i, 0], X[y == i, 1], X[y == i, 2], marker=markers,
c=scatter_colors, label=f'Class {i}')
ax.legend(frameon=False)
#3.调用LDA算法进行降维，并拟合数据
lda = LinearDiscriminantAnalysis(n_components=2)
X_new = lda.fit_transform(X, y)
plt.subplot(122)
for i in range(3):
    markers = shapes[i]
    scatter_colors = colors[i]
    plt.scatter(X_new[y == i, 0], X_new[y == i, 1], marker=markers,
c=scatter_colors, label=f'Class {i}')
plt.legend(frameon=False)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.tight_layout()
```

```
plt.show()
```



由结果可以看出降维后的二维空间可以清晰地对三个类别的数据进行区分。因此，LDA算法能够找到一个投影方向，使得不同类别的数据点在投影后能有很好的区分性，利用类别信息来最大程度保留数据的判别性能。

1.8 t-SNE（t-分布随机邻居嵌入）

1.8.1基本思想

t-SNE（t-Distributed Stochastic Neighbor Embedding）是一种**非线性降维算法**，用于**高维数据的可视化**。
其核心思想是：

- **在高维空间中，相似的数据点应该在低维空间中保持相似**；
- **在低维空间中，相异的数据点应该远离。**

1.8.2原理推导

- 适用于**高维数据可视化**，通过**保持数据局部结构**，将数据映射到 2D/3D 空间。
- 在高维空间中，样本*i*与*j*之间的**相似度（条件概率）** 定义为：

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}$$

其中σi是高斯分布的标准差（通过 perplexity 参数确定）。

- 在低维空间，使用**t-分布**定义样本之间的相似度：

$$q_{j|i} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_k\|^2)^{-1}}$$

- 目标是最小化**KL 散度**（衡量两个分布的差异）：

$$KL(P \parallel Q) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

通过**梯度下降** 优化，调整yi的位置，使 KL 散度最小。

t-SNE 通过 **梯度下降来优化目标函数，使得** 低维表示yi逼近高维数据的结构**。
梯度更新规则：

$$\Delta y_i = \eta \sum_{j \neq i} (p_{ij} - q_{ij})(y_i - y_j)$$

其中：

- η是学习率；
- 由于 t-分布的长尾特性，远离的点不会被拉得过近，避免 "局部挤压"。

1.8.3算法评价

特点：

保留局部结构：相似的数据点在低维空间中仍然相似；
克服"拥挤问题"：使用 t-分布，使远离的点更分散；
适用于非线性数据，不像 PCA 只能处理线性结构。

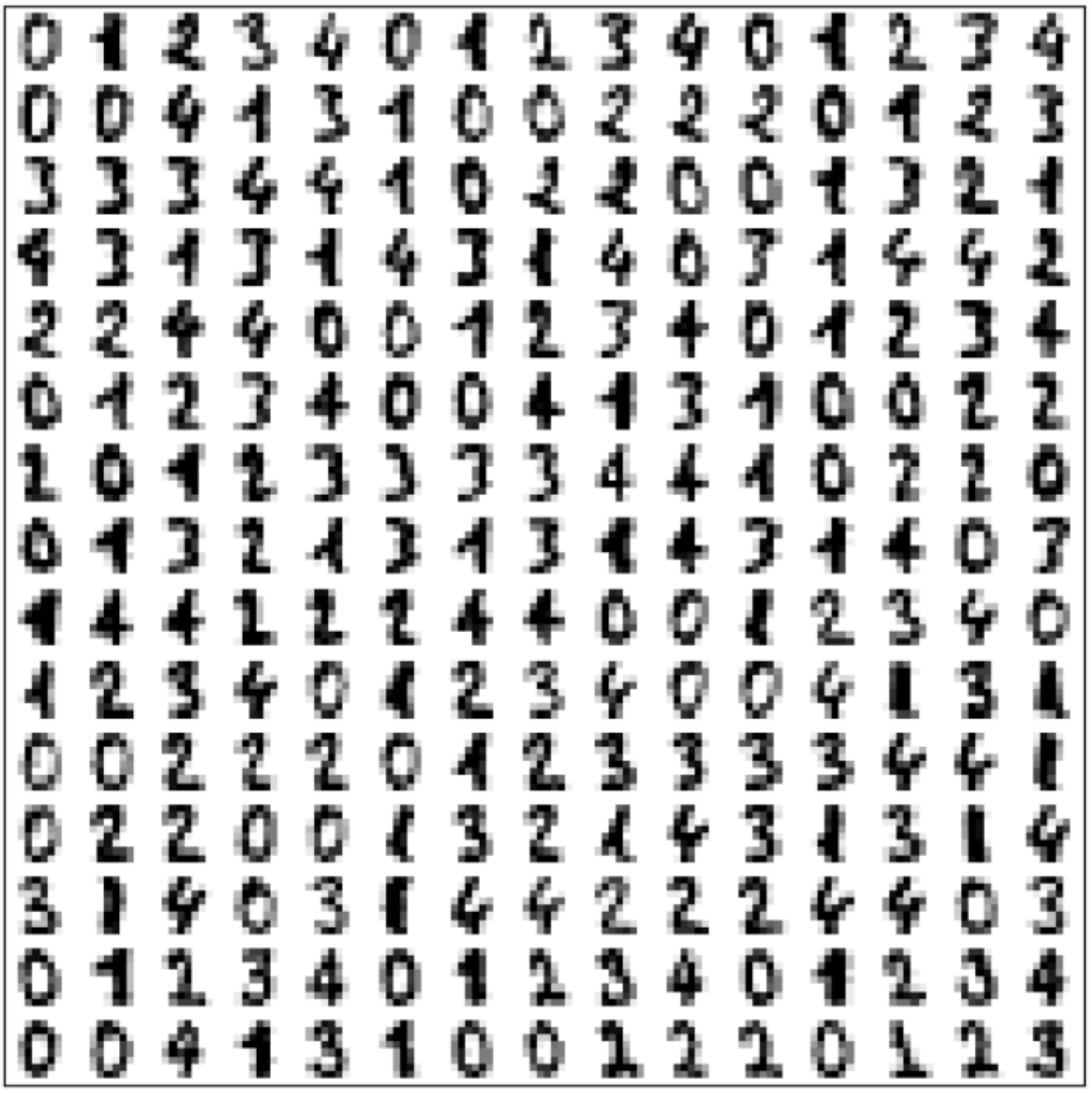
局限性：
高时间复杂度：O(N2)，不适合大规模数据；
随机初始化导致结果不稳定，同一数据集多次运行可能得到不同结果；
无法用于新数据映射，不像 PCA 可以直接投影新样本。

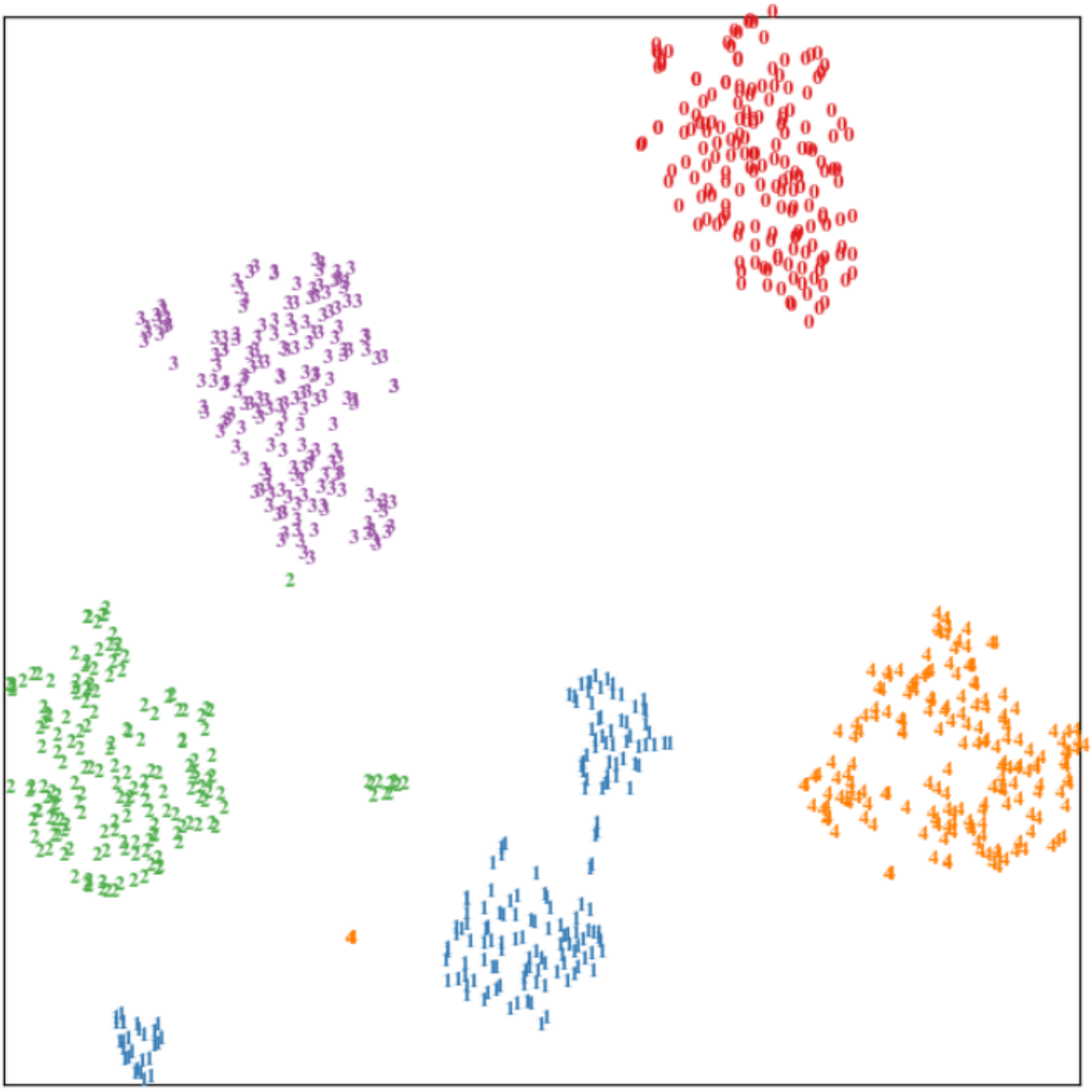
1.8.4实践案例：基于t-SNE的手写数字数据集降维

本案例使用t-SNE算法对手写数字图片数据集进行降维，共有1797张分辨率为8×8像素的手写数字图片，包含0到9十类数字，本案例从中选取0到4五类数字。每个图象可被展平为一个长度为64的特征向量，t-SNE算法根据这些像素特征计算样本之间相似度，将其转化为二维空间坐标。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import manifold, datasets
#1.构建数据集
digits = datasets.load_digits(n_class=5)
X, y = digits.data, digits.target
n_samples, n_features = X.shape
n = 15
img = np.zeros((10 * n, 10 * n))
#2.可视化原始数据集
for i in range(n):
    ix = 10 * i + 1
    for j in range(n):
        iy = 10 * j + 1
        img[ix:ix + 8 , iy:iy +8 ] = X[i * n + j].reshape((8,8))
plt.figure(figsize=(8,8))
plt.imshow(img, cmap=plt.cm.binary)
plt.xticks([])
plt.yticks([])
plt.show()
# 3.调用TNE函数，拟合数据
tsne = manifold.TSNE(n_components=2, random_state=501)
X_tsne = tsne.fit_transform(X)
x_min, x_max = X_tsne.min(0), X_tsne.max(0)
X_norm = (X_tsne - x_min) / (x_max - x_min)
```

```
#4.可视化降维结果
plt.figure(figsize=(8,8))
for i in range(X_norm.shape[0]):
    plt.text(X_norm[i,0], X_norm[i,1], str(y[i]), color=plt.cm.Set1(y[i]),
fontdict={'weight': 'bold', 'size': 9})
plt.xticks([])
plt.yticks([])
plt.show()
```





通过观察t-SNE算法降维后的结果，发现二维空间中具有相似特征的数据点形成紧密的簇，不同类别的数据簇相隔较远，这表明t-SNE能有效区分不同类别数据点，提高数据的可分离性。

1.9 UMAP（Uniform Manifold Approximation and Projection）

1.9.1基本思想

UMAP（统一流形逼近与投影）是一种基于 **流形学习（Manifold Learning）**的**非线性降维方法**，旨在将高维数据映射到低维空间，同时最大程度地保留数据的局部和全局结构。

UMAP 的核心思想基于流形假设：**高维数据实际上位于一个低维流形上**，我们可以通过数学方法揭示这一低维结构，并将其嵌入到更易解释的低维空间（如 2D 或 3D）。

具体来说，UMAP 通过以下三个核心步骤实现降维：①构建高维数据的拓扑结构，**即创建数据点之间的加权图，以捕捉数据的局部结构。**②在低维空间中重构拓扑结构，**以保持数据点之间的关系。** ③使用优化方法调整低维嵌入**，确保数据的局部和全局结构尽可能与高维一致。

1.9.2数学推导

UMAP 主要由三个核心步骤组成：

(1) 构建高维空间的拓扑结构

在高维数据空间，UMAP 认为数据分布在某个**低维流形（manifold** 上。它的目标是：

- 估计数据点之间的局部相似性；
- 用 **加权图（weighted graph）** 进行表示。

(a) 计算局部连接概率

对于高维空间中的数据点xi和xj，UMAP 定义了一个邻域半径ρi和尺度参数σi，并用**基于高斯核的模糊相似度**来计算点对之间的连接概率：

$$p_{ij} = \exp\left(-\frac{\max(0, d(x_i, x_j) - \rho_i)}{\sigma_i}\right)$$

其中：

- $$d(x_i, x_j)$$

  是欧氏距离；
- ρi控制最小距离（保证每个点至少有一个最近邻）；
- σi 是局部尺度参数，通过平衡最近邻个数来计算。

邻域大小自适应调整：

- **密集区域**：σi小，使相邻点的相似性更高；
- **稀疏区域**：σi大，使远离的点仍然保持相似性。

最终，我们构建了一个 **加权无向图**，其中边权重pij反映了数据点之间的连接强度。

---

(2) 低维空间映射

在低维空间（如 2D 或 3D），UMAP 采用 **模糊集合交叉熵** 进行优化，使高维结构在低维中尽可能保持。

(a) 定义低维相似度

在低维空间（嵌入空间），使用 **t-分布替代高斯分布**，定义新的相似度：

$$qij = \frac{1}{1 + a \cdot d(y_i, y_j)^b}$$

其中：

- d(yi,yj)是低维空间的欧氏距离；
- a和 b是优化出的参数，用于调整相似度分布形态。

这种 **t-分布形式** 允许远离的数据点保持更大的间隔，防止数据 "挤在一起"。

---

(3) 目标函数与优化

UMAP 通过 **最小化高维和低维相似度的交叉熵** 进行优化：

$$C \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} + (1 - p_{ij}) \log \frac{1 - p_{ij}}{1 - q_{ij}}$$

其中：

- 第一项：**鼓励高维相似点在低维中靠近**；
- 第二项：**鼓励高维不相似的点在低维空间远离。**

为了加速计算，UMAP 采用 **负采样（negative sampling）** 进行优化，使远离的数据点拉开的效果更好。

1.9.3算法评价

- **保持局部和全局信息**：UMAP 不仅保留局部邻域关系，还能更好地保存全局数据结构。
- **计算速度快，适用于大规模数据**：UMAP 采用近似 K 近邻搜索和高效的优化方法，计算复杂度为 O(NlogN)O(N \log N)O(NlogN)，相比传统方法更高效。
- **可扩展到流式数据**：UMAP 可以学习一个低维嵌入映射，用于快速投影新数据点，而不需要重新训练整个模型。

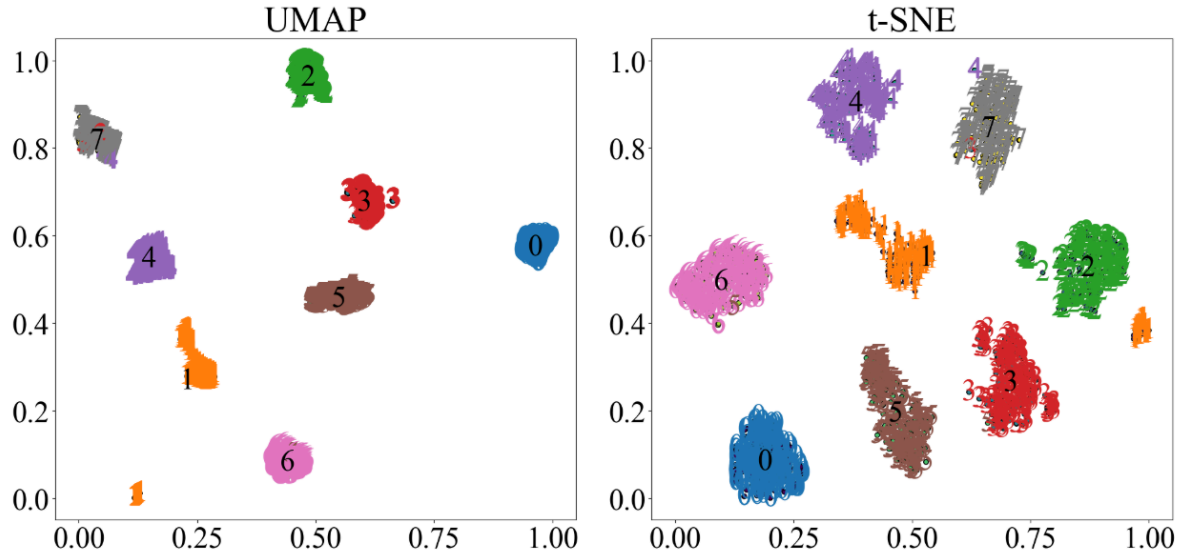
UMAP 适用于 **高维数据可视化、特征降维、聚类前处理**，并广泛应用于 **图像处理、自然语言处理、生物信息学（基因数据）** 等领域。

1.9.4实践案例：基于UMAP的手写数字数据集降维

本案例使用UMAP算法对手写数据集进行降维

```
from umap import UMAP
from sklearn.manifold import TSNE
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_digits
import numpy as np
import matplotlib.pyplot as plt
# 1. 生成数据集
digits = load_digits(n_class=8)
X, y = digits.data, digits.target
n_samples, n_features = X.shape
n = 5000
X, y = X[:n], y[:n]
# 2. 调用UMAP算法将数据集降至二维
reducer_umap = UMAP(n_components=2, random_state=0)
embedding_umap = reducer_umap.fit_transform(X)
scaler_umap = MinMaxScaler()
embedding_umap = scaler_umap.fit_transform(embedding_umap)
# 3. 调用t-SNE算法将数据集降至二维
reducer_tsne = TSNE(n_components=2, random_state=0)
embedding_tsne = reducer_tsne.fit_transform(X)
scaler_tsne = MinMaxScaler()
embedding_tsne = scaler_tsne.fit_transform(embedding_tsne)
# 4. 可视化UMAP算法结果
plt.rcParams['font.sans-serif']=['Times New Roman']
plt.rcParams.update({'font.size': 35})
```

```
fig = plt.figure(figsize=(20, 10))
plt.subplot(1, 2, 1)
plt.title('UMAP')
for i in range(embedding_umap.shape[0]):
 plt.text(
 embedding_umap[i, 0],
 embedding_umap[i, 1],
 str(y[i]),
 color=plt.cm.tab10(y[i]),
 fontdict={"weight": "bold"},
 va="center",
 ha="center",
)
plt.scatter(embedding_umap[:, 0], embedding_umap[:, 1], c=y, cmap='viridis',
s=30, edgecolors='k')
for digit in range(len(np.unique(y))):
 digit_indices = np.where(y == digit)
 plt.text(
 np.mean(embedding_umap[digit_indices, 0]),
 np.mean(embedding_umap[digit_indices, 1]),
 str(digit),
 color='black',
 va="center",
 ha="center",
)
plt.tick_params(axis='both', which='major')
# 5. 可视化t-SNE算法结果
plt.subplot(1, 2, 2)
plt.title('t-SNE')
for i in range(embedding_tsne.shape[0]):
 plt.text(
 embedding_tsne[i, 0],
 embedding_tsne[i, 1],
 str(y[i]),
 color=plt.cm.tab10(y[i]),
 va="center",
 ha="center",
)
plt.scatter(embedding_tsne[:, 0], embedding_tsne[:, 1], c=y, cmap='viridis',
s=30, edgecolors='k')
for digit in range(len(np.unique(y))):
 digit_indices = np.where(y == digit)
 plt.text(
 np.mean(embedding_tsne[digit_indices, 0]),
 np.mean(embedding_tsne[digit_indices, 1]),
 str(digit),
 color='black',
 va="center",
 ha="center",
)
plt.tick_params(axis='both', which='major')
plt.tight_layout()
plt.show()
```



此代码使用 **UMAP和 t-SNE**两种非线性降维方法，将 **scikit-learn** 提供的 **手写数字数据集** 从高维降至二维，并进行可视化。

首先，代码使用 `load_digits(n_class=8)` 载入 0-7 类的手写数字数据，每个样本是一张 **8×8 的灰度图像**，然后选取最多 5000 个样本用于降维。接着，利用 **UMAP** 进行降维，使用 `UMAP(n_components=2, random_state=0)` 计算二维投影，并通过 `MinMaxScaler` 进行归一化，使数据范围缩放到 `[0,1]`。同样，代码使用 **t-SNE** 进行降维，并应用相同的归一化操作。

在可视化部分，代码使用 `matplotlib` 绘制 UMAP 和 t-SNE 结果的 **散点图**，并使用 `plt.text` 在每个数据点上标注其类别标签（即对应的数字 0-7）。此外，还计算每个类别的中心位置，并在图中标注，以便更清楚地观察不同类别的分布。最终，代码以 `plt.show()` 展示结果，使得 UMAP 和 t-SNE 在同一张图上进行对比，以评估它们在降维过程中保留数据结构的能力。

由运行结果可以看出，UMAP算法和t-SNE算法均可将不同类别数字有效分离，但是相比于t-SNE算法，UMAP算法形成的簇间距离更大，簇内距离更小，这展示了UMAP算法优秀的分类效果。